ĐẠI HỌC QUỐC GIA THÀNH PHỐ HỒ CHÍ MINH TRƯỜNG ĐẠI HỌC KHOA HỌC TỰ NHIỀN KHOA CÔNG NGHỆ THÔNG TIN



ĐỒ ÁN 01 MÔN KIẾN TRÚC MÁY TÍNH VÀ HỢP NGỮ

Đề tài:

BIỂU DIỄN VÀ TÍNH TOÁN SỐ HỌC TRONG MÁY TÍNH

Người thực hiện:

Nguyễn Hoàng Thái Dương (18120336)

Trần Thanh Quang (18120230)

Trương Trọng Lộc (18120197)

Giảng viên hướng dẫn:

Thầy ThS. Lê Viết Long

Thành phố Hồ Chí Minh – Tháng 05, 2020

MŲC LŲC

MỤC LỤC	2
1. THÀNH VIÊN VÀ PHÂN CHIA CÔNG VIỆC:	5
2. BIỂU DIỄN SỐ NGUYÊN LỚN 128 BIT	5
2.1. Thiết kế cấu trúc dữ liệu QInt:	5
2.2. Chuyển đổi giữa các hệ cơ số:	5
2.2.1. Ý tưởng chuyển đổi:	5
2.2.2. Nhóm hàm SetBit, GetBit:	5
2.2.3. Chuyển đổi từ chuỗi nhị phân (string) sang QInt:	6
2.2.4. Chuyển đổi từ chuỗi thập phân (string) sang QInt:	7
2.2.5. Chuyển đổi từ chuỗi thập lục phân (string) sang QInt:	7
2.2.6. Chuyển đổi từ chuỗi ở hệ cơ số 2, 10, 16 (string) sang QInt:	7
2.2.7. Chuyển đổi từ QInt sang chuỗi nhị phân (string):	7
2.2.8. Chuyển đổi từ QInt sang chuỗi thập phân (string):	7
2.2.9. Chuyển đổi từ QInt sang chuỗi thập lục phân (string):	8
2.2.10. Chuyển đổi từ QInt sang chuỗi ở hệ cơ số 2, 10, 16 (string):	8
2.2.11. Chuyển đổi từ chuỗi ở hệ cơ số a sang chuỗi ở hệ cơ số b (a, b = 2, 10, 16):	8
2.3. Các nhóm toán tử:	8
2.3.1. Toán tử cộng:	8
2.3.2. Toán tử trừ:	9
2.3.3. Toán tử nhân:	9
2.3.4. Toán tử chia và lấy dư:	9
2.3.5. Toán tử so sánh và gán:	.11

2.3.6. Toán tử AND, OR, XOR, NOT:	11
2.3.7. Toán tử dịch trái, dịch phải:	11
2.3.8. Toán tử xoay trái, xoay phải:	12
3. BIỂU DIỄN SỐ CHẨM ĐỘNG CHÍNH XÁC CAO 128 BIT	12
3.1. Thiết kế kiểu dữ liệu QFLoat:	12
3.2. Chuyển đổi giữa các hệ cơ số:	12
3.2.1. Ý tưởng chuyển đổi:	12
3.2.2. Nhóm hàm SetBit, GetBit:	12
3.2.3. Nhóm hàm kiểm tra các trường hợp đặc biệt:	12
3.2.4. Chuyển đổi từ chuỗi nhị phân (string) sang QFloat:	13
3.2.5. Chuyển đổi từ chuỗi thập phân (string) sang QFloat:	13
3.2.6. Chuyển đổi từ chuỗi ở hệ cơ số 2, 10 (string) sang QFloat:	14
3.2.7. Chuyển đổi từ QFloat sang chuỗi nhị phân (string):	14
3.2.8. Chuyển đổi từ QFloat sang chuỗi thập phân (string):	15
3.2.9. Chuyển đổi từ QFloat sang chuỗi ở hệ cơ số 2, 10 (string):	15
3.3. Các nhóm toán tử:	16
3.3.1. Toán tử cộng:	16
3.3.2. Toán tử trừ:	16
3.3.3. Toán tử nhân:	17
3.3.4. Toán tử chia:	17
4. PHẠM VI BIỂU DIỄN CỦA CÁC KIỂU DỮ LIỆU ĐÃ THIẾT KẾ	18
4.1. Kiểu số nguyên lớn QInt 128 bit:	18
4.2. Kiểu số chấm động chính xác cao:	
5. GIAO DIỆN NGƯỜI DÙNG	

5.1. Chế độ QInt:		19
5.2. Chế độ QFloat:		20
6. GIAO DIÊN CHƯƠNG	G TRÌNH ỨNG VỚI TEST CASE	20
6.1. Tham số dòng lệr	ıh:	20
6.1.1. Kiểu dữ liệu Q	<u>I</u> nt:	20
6.1.2. Kiểu dữ liệu Q	Ploat:	25
6.2. Giao diện người d	dùng:	31
7. ĐÁNH GIÁ MÚC ĐỘ	HOÀN THÀNH	32
8. TÀI LIÊU THAM KHA	ÅO	33

1. THÀNH VIÊN VÀ PHÂN CHIA CÔNG VIỆC:

STT	MSSV	Họ và tên	Công việc		
1	18120336	Nguyễn Hoàng Thái	Giao diện Calculator, phép nhân, chia		
1	16120330	Dương	QFloat.		
			Chuẩn bị 296 bộ testcase, nhập xuất		
2	18120230	Trần Thanh Quang	QInt, QFloat, các toán tử so sánh, AND		
Δ	18120230		OR, XOR, NOT, phép dịch trái, dịch		
			phải, xoay trái, xoay phải QInt.		
			Chuyển đổi các hệ cơ số QInt và QFloat,		
3	18120197	Trương Trọng Lộc	toán tử cộng, trừ, nhân, chia, chia lấy dư		
			QInt và cộng, trừ QFloat.		

2. BIỂU DIỄN SỐ NGUYÊN LỚN 128 BIT

2.1. Thiết kế cấu trúc dữ liệu QInt:

- Số nguyên lớn QInt bao gồm 128 bit được lưu trữ bởi mảng *data* gồm 4 phần tử kiểu số nguyên. Ở phương thức khởi tao mặc đinh, tất cả 128 bit này đều là bit 0.
- Vì 1 phần tử kiểu số nguyên có kích thước 4 bytes nên ta sử dụng 4 phần tử để biểu diễn (thỏa yêu cầu 16 bytes như đề bài).

2.2. Chuyển đổi giữa các hệ cơ số:

2.2.1. Ý tưởng chuyển đổi:

Đầu vào sẽ là một chuỗi String ở dạng nhị phân / thập lục phân / thập phân. Sau đó ta lưu chuỗi String này vào QInt bằng các hàm chuyển đổi. Từ QInt ta chuyển sang lại dạng chuỗi ở các dạng nhị phân / thập lục phân / thập phân. Như vậy, ta có thể chuyển từ chuỗi String từ hệ cơ số a sang chuỗi String ở hệ cơ số b dựa vào trung gian là QInt.

2.2.2. Nhóm hàm SetBit, GetBit:

- Lưu ý về việc lưu các dãy bit. Ta xét ví dụ sau: Giả sử ta có chuỗi 8 bit. Khi đó, đối với dạng chuỗi (string), vị trí từng bit, lúc này mỗi bit là phần tử của string nên sẽ được

đánh số thứ tự từ trái sang phải. Tuy nhiên, khi lưu trữ dãy bit, ta đánh số từ phải sang trái. Giả sử ta lưu dãy 8 bit này vào 1 mảng gồm 2 phần tử có kích thước 4 bit, ta lưu:

Vị trí bit (dạng string)	0	1	2	3	4	5	6	7
Vị trí bit (khi lưu trữ)	7	6	5	4	3	2	1	0
Mảng 2 phần tử (mỗi phần tử 4 bit)	A[0]		A[1]					
Vị trí bit ở mỗi phần tử trong mảng	3	2	1	0	3	2	1	0

- Để dễ hiểu, ta xét ví dụ, dãy 8 bit 1100 1011 ở dạng chuỗi (string), khi đó phần tử str[0] = '1' sẽ tương ứng với vị trí bit thứ 7 khi lưu trữ. Khi đó bit này sẽ được lưu trữ vào A[1] ở vị trí thứ 3. Sở dĩ là A[1] vì bit lưu trữ ở vị trí thứ 7 chia cho kích thước một phần tử là 4, lấy phần nguyên ta được 1; ở vị trí thứ 3 trong A[1] thì 7 chia 4 lấy phần dư là 3. Từ đó, ta thấy với dãy 1100 1011 thì A[0] gồm dãy 4 bit 1011 và A[1] gồm dãy 4 bit 1100.
- Tổng quát để xác định vị trí bit lưu ở đâu trong QInt dựa vào vị trí bit khi lưu trữ, ta làm như sau:
 - Xác định block dựa vào vị trí bit khi lưu trữ bằng cách lấy vị trí đó chia cho
 kích thước của một block. Với QInt, mỗi block là kiểu số nguyên nên kích
 thước là 32 bit.
 - Xác định vị trí bit trong block bằng cách lấy vị trí bit khi lưu trữ chia lấy dư cho kích thước của một block.
- Sau khi xác định được vị trí bit trong QInt, ta tiến hành GetBit hoặc SetBit như sau:
 - O Với GetBit, ta thực hiện phép AND với 1 tại bit đó.
 - Với SetBit, nếu là bit 1, ta thực hiện phép OR với 1 tại bit đó (vì OR với 1 sẽ ra 1); nếu là bit 0, ta thực hiện phép AND với 0 tại bit đó (vì AND với 0 sẽ ra 0)

2.2.3. Chuyển đổi từ chuỗi nhị phân (string) sang QInt:

Ta lần lượt duyệt qua từng phần tử của chuỗi string. Cứ mỗi lần duyệt ta gọi hàm SetBit để lưu vào QInt. Ở đây, ta duyệt từ đầu đến cuối của chuỗi string, nên khi SetBit vị trí sẽ là $\operatorname{str.length}() - 1 - i$, với i là biến đếm duyệt chuỗi string, $\operatorname{str.length}()$ là độ dài chuỗi string (vì vị trí set bit là vị trị lưu trữ - sẽ được đánh số thứ tự ngược với chuỗi string như đã trình bày ở trên).

2.2.4. Chuyển đổi từ chuỗi thập phân (string) sang QInt:

- Đầu tiên ta xét chuỗi thập phân biểu thị số âm hay dương, nếu có dấu '-' ta thực hiện xóa dấu này ra khỏi chuỗi và có lưu lại sự thay đổi này bởi biến sign kiểu bool để làm cơ sở xét dấu lúc sau.
- Ta thực hiện chuyển chuỗi thập phân sang QInt bằng cách chia chuỗi này cho 2, phần dư sẽ được SetBit vào QInt, còn phần nguyên ta tiếp tục chia cho 2. Lặp lại như vậy cho đến khi chuỗi bằng "0" hoặc chuỗi rỗng.
- Ta dựa vào tình trạng biến *sign*, nếu biến *sign* biểu thị số âm ta thực hiện chuyển sang dạng bù 2 (ta thực hiện đổi bit kết quả tìm được ở trên, sau đó cộng cho 1).

2.2.5. Chuyển đổi từ chuỗi thập lục phân (string) sang QInt:

- Do ở hệ thập lục phân, biểu diễn bởi 16 ký tự là 0 đến 9 và A đến F. Nên trước tiên ta thực hiện chuẩn hóa các ký tự đầu vào thành kí tự in hoa (nếu đầu vào là chữ thường).
- Điểm đặc biệt ở hệ thập lục phân là mỗi ký tự biểu diễn của nó sẽ được biểu diễn bởi 4 bit nhị phân (Vì mỗi ký tự của nó chạy từ 0 đến 15). Vì vậy ta sẽ duyệt lần lượt từng ký tự ở hệ thập lục phân và chuyển nó về dạng 4 bit nhị phân.
- Lưu ý khi chuyển, nếu gặp ký tự chữ ta phải chuyển đó về dạng số tương ứng rồi sau đó mới chuyển về dạng 4 bit. Cụ thể A là 10, B là 11, C là 12, D là 13, E là 14 và F là 15.

2.2.6. Chuyển đổi từ chuỗi ở hệ cơ số 2, 10, 16 (string) sang QInt:

Ta sử dụng câu lệnh *if* trên tham số đầu vào (đầu vào sẽ được truyền vào biểu thị hệ cơ số cần được chuyển đổi vào QInt). Sau đó dựa vào đầu vào ta lần lượt gọi các hàm đã trình bày ở 2.2.3, 2.2.4 và 2.2.5 để lưu trữ vào QInt.

2.2.7. Chuyển đổi từ QInt sang chuỗi nhị phân (string):

Ta thực hiện lấy từng bit trong QInt và cộng vào chuỗi kết quả. Sau đó ta loại bỏ các số 0 thừa ở đầu. Lưu ý giữ lại 1 bit 0 nếu 128 bit đều là 0.

2.2.8. Chuyển đổi từ QInt sang chuỗi thập phân (string):

- Ta thực hiện lấy từng bit trong QInt theo đúng thứ tự lưu trữ (từ 0 đến 126, trừ bit dấu). Tại mỗi bit nếu giá trị bit là 1 ta thực hiện cộng với 2^hi với i là số thứ tự của bit. - Ta xét bit dấu, nếu bit dấu là 1, ta thực hiện phép trừ giữa 2^127 và kết quả tìm được ở trên, sau đó thêm '-' vào trước chuỗi kết quả.

2.2.9. Chuyển đổi từ QInt sang chuỗi thập lục phân (string):

- Đầu tiên, ta tạo kiểu dữ liệu map để lưu trữ các ký tự biểu diễn hệ thập lục phân ở dạng
 4 bit. Sở dĩ chọn kiểu dữ liệu map vì nó dễ dàng truy xuất phần tử với index dạng chuỗi.
- Sau đó ta duyệt lần lượt bộ 4 bit trong QInt từ vị trí i = 0 và dựa vào chuỗi 4 bit đó ta cộng vào chuỗi kết quả ký tự ở hệ thập lục phân tương ứng thông qua kiểu dữ liệu map.
- Sau đó ta thực hiện xóa các số 0 thừa (nếu có) và thực hiện đảo ngược lại kết quả (vì i
 = 0 nên ta phải đảo ngược kết quả).

2.2.10. Chuyển đổi từ QInt sang chuỗi ở hệ cơ số 2, 10, 16 (string):

Ta sử dụng câu lệnh *if* trên tham số đầu vào (đầu vào sẽ được truyền vào biểu thị hệ cơ số cần được chuyển đổi ra string từ QInt). Sau đó dựa vào đầu vào ta lần lượt gọi các hàm đã trình bày ở 2.2.7, 2.2.8 và 2.2.9 để chuyển từ QInt sang string.

2.2.11. Chuyển đổi từ chuỗi ở hệ cơ số a sang chuỗi ở hệ cơ số b (a, b = 2, 10, 16):

Ta sử dụng câu lệnh if trên tham số đầu vào (đầu vào sẽ được truyền vào biểu thị hệ cơ số a – hệ cơ số cần phải chuyển đổi và hệ cơ số b – sau khi chuyển đổi). Sau đó dựa vào đầu vào ta lần lượt gọi các hàm đã trình bày ở $\underline{2.2.6}$ và $\underline{2.2.10}$ để thực hiện việc chuyển đổi.

2.3. Các nhóm toán tử:

2.3.1. Toán tử cộng:

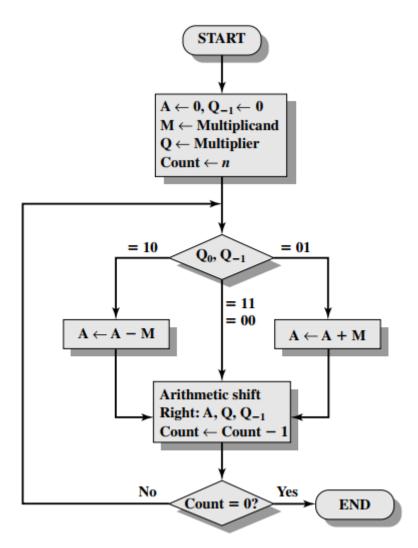
Phép cộng 2 số QInt được thực hiện dựa trên nguyên tắc cộng bit. Ta cộng lần lượt các bit theo thứ tự từ phải sang trái. Trong quá trình cộng có biến *carry* để lưu lại bit nhớ. Ta cộng bit ở số QInt trên với bit tương ứng ở số QInt dưới với bit nhớ ta được kết quả lưu ở biến *tmp*, sau đó ta thực hiện SetBit bằng cách lấy kết quả vừa tìm lấy dư cho 2, đồng thời cập nhật biến nhớ bằng cách lấy *tmp* chia cho 2.

2.3.2. Toán tử trừ:

Ta thực hiện chuyển số trừ sang dạng bù 2 (đổi bit và cộng 1), sau đó thực hiện phép cộng giữa số bị trừ và số trừ ở dạng bù 2.

2.3.3. Toán tử nhân:

Ta thực hiện theo thuật toán Booth. Lưu đồ thuật toán Booth sau đây được trích từ Chương 9, sách Computer Organization and Architecture Designing for performance (8th edition) của William Stallings.



2.3.4. Toán tử chia và lấy dư:

- Phép chia và lấy dư được thực hiện theo thuật toán chia không dấu. Nếu số bị chia hay số chia là số âm, ta thực hiện chuyển các số âm đó sang dạng bù 2 để thực hiện chia

- không dấu. Sau khi chia không dấu ta dựa vào dấu của số bị chia và số chia để xác định dấu của thương và số dư cho phù hợp.
- Ta lưu ý về thương và số dư. Gọi D là số bị chia, V là số chia, Q là thương, R là số dư. Ta có biểu thức sau: D = Q × V + R. Dấu của R chính là dấu của D. Dấu của Q chính là dấu của D nhân với dấu của V. Xét ví dụ sau để hiểu rõ:

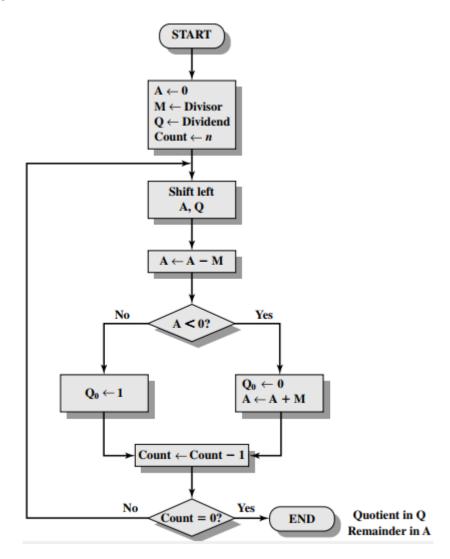
o
$$D = 7$$
, $V = 3 \Rightarrow Q = 2$, $R = 1$

o
$$D = -7$$
, $V = 3 \Rightarrow Q = -2$, $R = -1$

o
$$D = 7$$
, $V = -3 \Rightarrow Q = -2$, $R = 1$

$$O$$
 D = -7, V = -3 => Q = 2, R = -1

Lưu đồ thuật toán chia không dấu sau đây được trích từ Chương 9, sách Computer
 Organization and Architecture Designing for performance (8th edition) của William Stallings.



2.3.5. Toán tử so sánh và gán:

- Phép >: Kiểm tra xem 2 số có cùng dấu hay không, nếu khác dấu, ta kiểm tra số đầu tiên nếu là âm sẽ return false, ngược lại return true. Nếu cùng dấu, ta lần lượt so sánh các bit theo thứ tự từ bit kế bit dấu đến bit thứ 0 trong vị trí lưu trữ. Nếu bit tại vị trí *i* ở số thứ nhất nhỏ hơn bit ở số thứ 2, ta trả về bit dấu của số thứ nhất. Nếu bit bit tại vị trí *i* ở số thứ 1 lớn hơn bit ở số thứ 2, ta đổi bit dấu của số thứ nhất (lấy NOT) và trả về.
- Phép ==: Nếu hai số có bit nào khác nhau thì trả về false, ngược lại trả về true.
- Phép !=: Nếu hai số không bằng nhau thì return true, ngược lại return false.
- Phép <: Nếu số thứ nhất không lớn hơn và không bằng số thứ 2 return true; ngược lại return false
- Phép >=: Nếu số thứ nhất không nhỏ hơn số thứ 2 thì return true, ngược lại return false.
- Phép <=: Nếu số thứ nhất không lớn hơn số thứ 2 thì return true, ngược lại return false.
- Toán tử gán =: Nếu 2 địa chỉ của 2 biến khác nhau ta thực hiện gán thuộc tính của biến này sang biến khác.

2.3.6. Toán tử AND, OR, XOR, NOT:

- Ở các toán tử AND, OR, XOR, ta thực hiện trên từng cặp bit của 2 số QInt.
- Ở toán tử NOT, ta đổi từng bit của số đó.
- Cụ thể:
 - O' phép AND, nếu cả 2 bit của 2 số QInt là 1 ta SetBit 1 vào số QInt kết quả, ngược lại SetBit 0.
 - O phép OR, nếu cả 2 bit của 2 số QInt là 0 ta SetBit 0 vào số QInt kết quả, ngược lại SetBit 1.
 - O phép XOR, nếu 2 bit của 2 số QInt bằng nhau, ta SetBit 0 vào số QInt kết quat, ngược lại SetBit 1.
 - o Ở phép NOT, ta thực hiện đổi từng bit (từ 0 sang 1 và từ 1 sang 0).

2.3.7. Toán tử dịch trái, dịch phải:

- Ta truyền vào tham số là số lượng bit cần dịch.
- Ở đây ta thực hiện phép dịch số học. Khi dịch trái ta thêm các bit 0 vào cuối dãy bit sau khi dịch. Khi dịch phải ta thêm các bit dấu vào đầu dãy bit sau khi dịch.

2.3.8. Toán tử xoay trái, xoay phải:

- Ta truyền vào tham số là số lượng bit cần xoay.
- Khi xoay trái / xoay phải x bit, ta lấy số lượng bit cần xoay bằng cách chia lấy dư với 128 (vì phép xoay sẽ thực hiện tuần hoàn với chu kỳ 128 bit).
- Mỗi lần xoay ta dịch trái / dịch phải 1 bit, lặp lại cho đến khi đủ x lần.

3. BIỂU DIỄN SỐ CHẨM ĐỘNG CHÍNH XÁC CAO 128 BIT

3.1. Thiết kế kiểu dữ liệu QFLoat:

- Số chấm động chính xác cao QFloat bao gồm 128 bit được lưu trữ bởi mảng data gồm 4 phần tử kiểu số nguyên. Ở phương thức khởi tạo mặc định, tất cả 128 bit này đều là bit 0.
- Vì 1 phần tử kiểu số nguyên có kích thước 4 bytes nên ta sử dụng 4 phần tử để biểu diễn (thỏa yêu cầu 16 bytes như đề bài).
- Trong 128 bit, có 1 bit biểu diễn dấu, 15 bit biểu diễn phần mũ và 112 bit biểu diễn phần định trị.

3.2. Chuyển đổi giữa các hệ cơ số:

3.2.1. Ý tưởng chuyển đổi:

Đầu vào sẽ là một chuỗi String ở dạng nhị phân / thập phân. Sau đó ta lưu chuỗi String này vào QFloat bằng các hàm chuyển đổi. Từ QFloat ta chuyển sang lại dạng chuỗi ở các dạng nhị phân / thập phân. Như vậy, ta có thể chuyển từ chuỗi String từ hệ cơ số a sang chuỗi String ở hệ cơ số b dựa vào trung gian là QFloat.

3.2.2. Nhóm hàm SetBit, GetBit:

Ý tưởng và phương pháp thực hiện giống ở mục 2.2.2.

3.2.3. Nhóm hàm kiểm tra các trường hợp đặc biệt:

bool CheckZero() const;: Hàm kiểm tra có phải QFloat là 0 hay không. Nếu có bất kỳ
 bit nào trong 127 bit (trừ bit dấu) khác bit 0, return false.

- bool CheckDenormalized() const;: Hàm kiểm tra dạng số không chuẩn. Số không chuẩn là số có phần mũ toàn bit 0, và phần trị khác 0.
- bool CheckNaN() const;: Hàm kiểm tra có phải là dạng NaN không. Dạng NaN khi phần mũ toàn bit 1, và phần trị khác 0.
- bool CheckInf() const;: Hàm kiểm tra có phải là dạng Infinity không. Dạng Infinity khi phần mũ toàn bit 1, và phần trị toàn bit 0.

3.2.4. Chuyển đổi từ chuỗi nhị phân (string) sang QFloat:

Ta lần lượt duyệt qua từng phần tử của chuỗi string. Cứ mỗi lần duyệt ta gọi hàm SetBit để lưu vào QFloat. Ở đây, ta duyệt từ đầu đến cuối của chuỗi string, nên khi SetBit vị trí sẽ là $\operatorname{str.length}()-1-i$, với i là biến đếm duyệt chuỗi string, $\operatorname{str.length}()$ là độ dài chuỗi string (vì vị trí set bit là vị trị lưu trữ - sẽ được đánh số thứ tự ngược với chuỗi string như đã trình bày ở trên).

3.2.5. Chuyển đổi từ chuỗi thập phân (string) sang QFloat:

- Đầu tiên, ta xét dạng chuỗi thập phân nhập vào có phải dạng "0.0" hoặc "0" hay không,
 nếu có ta trả về phương thức khởi tạo mặc định của QFloat.
- Ta xét chuỗi thập phân biểu thị số âm hay dương, nếu có dấu '-' ta thực hiện xóa dấu này ra khỏi chuỗi và ta thực hiện SetBit dấu vào kết quả. Lưu ý bit dấu là vị trí thứ 127 như đã trình bày ở những phần trên.
- Ta thực hiện phân tách phần nguyên và phần thập phân của chuỗi đầu vào nhờ việc tìm kiếm dấu "." trong chuỗi đầu vào. Tuy nhiên, một số trường hợp chuỗi đầu vào không có dấu ".", ta thực hiện thêm vào sau chuỗi thập phân đầu vào ".0".
- Ta thực hiện đổi phần nguyên và phần thập phân của chuỗi đầu vào thành dãy bit nhị phân. Lưu ý, khi đổi phần thập phân thành các dãy bit, ta có các trường hợp sau đây:
 - Nếu phần nguyên khác 0, nghĩa là số này có dạng 1.F, đây là dạng chuẩn, nên ta thực hiện lấy phần thập phân nhân cho 2, và lưu lại phần nguyên. Sau đó lặplại việc này cho đến khi đủ bit. Trong quá trình lặp lại, khi đạt được giá trị 1.0 ta thực hiện lưu lại tình trạng này để hỗ trợ việc làm tròn sau này. Đồng thời, giá trị phần mũ được tính như sau: exp = intPartBit.size() 1 + BIAS trong đó exp là giá trị phần mũ, intPartBit.size() là chiều dài dãy bit nhị phân của phần nguyên, BIAS trong trường hợp QFloat là 2^14 1 = 16383. Sở dĩ có cách tính

- này là vì phần nguyên khác 0 nên tối thiểu là length intPartBit = 1 nên số lần dời dấu chấm là intPartBit.size() 1.
- Nếu phần nguyên bằng 0, ta có 2 trường hợp: số chuẩn hóa được và số không thể chuẩn hóa. Ta cũng thực hiện lấy phần thập phân nhân cho 2, và lưu lại phần nguyên. Sau đó lặp lại việc này cho đến khi đạt được giá trị 1.0. exp được tính bằng BIAS trừ đi số lần lặp. Nếu số lần lặp để tìm được giá trị 1.0 nhỏ hơn BIAS thì đây là trường hợp số có thể chuẩn hóa được, ta tiếp tục thực hiện nhân phần thập phân cho 2 và lưu lại phần nguyên cho đến khi đủ bit. Ngược lại là dạng số không thể chuẩn hóa, ta thực hiện reset lại chuỗi nhị phân của phần thập phân và nhân phần thập phân cho 2 và lưu lại phần nguyên và tăng exp lên 1. Lặp lại cho đến khi exp = 0. Trong quá trình lặp lại, khi đạt được giá trị 1.0 ta thực hiện lưu lại tình trạng này để hỗ trợ việc làm tròn sau này.
- Nếu ta chưa tìm được giá trị 1.0 khi lặp lại việc nhân phần thập phân cho 2. Ta tiếp tục lặp lại việc này thêm 3 lần, để thực hiện việc làm tròn theo nguyên tắc *round to the nearest; ties to even*.
- Ta làm tròn phần nguyên: khi thập phân toàn bit 1, ta cộng 1 vào *exp* và set lại phần thập phân là 0. Ví dụ: 10.1111 -> làm tròn: 11.0000.
- Nếu tổng bit nhị phân phần nguyên và phần thập phân chưa đủ 112 bit ta tiến hành thêm các bit 0 cho đủ bit.
- Sau cùng, ta đổi *exp* sang dãy bit nhị phân và tiến hành SetBit cho *exp* và phần trị vào kết quả.

3.2.6. Chuyển đổi từ chuỗi ở hệ cơ số 2, 10 (string) sang QFloat:

Ta sử dụng câu lệnh *if* trên tham số đầu vào (đầu vào sẽ được truyền vào biểu thị hệ cơ số cần được chuyển đổi vào QFloat). Sau đó dựa vào đầu vào ta lần lượt gọi các hàm đã trình bày ở <u>3.2.4</u> và <u>3.2.5</u> để lưu trữ vào QFloat.

3.2.7. Chuyển đổi từ QFloat sang chuỗi nhị phân (string):

Ta thực hiện lấy từng bit trong QFloat và cộng vào chuỗi kết quả. Lưu ý thứ tự khi GetBit và cộng vào chuỗi. Ta thực hiện GetBit từ bit dấu (bit thứ 127) về bit thứ 0.

3.2.8. Chuyển đổi từ QFloat sang chuỗi thập phân (string):

- Ta kiểm tra QFloat có phải là 0 hay dạng NaN hay không, nếu phải ta thực hiện return chuỗi báo tương ứng.
- Ta lưu lại bit dấu của QFloat (bit thứ 127).
- Ta kiểm tra QFloat có phải dạng Infinity hay không, nếu phải ta xét bit dấu và return Inf hay -Inf cho phù hợp.
- Nếu QFloat không nằm trong các dạng đặc biệt trên, ta thực hiện:
 - Tách phần mũ và phần trị từ QFloat (ta chuyển QFloat sang chuỗi nhị phân cho dễ thao tác).
 - \circ Ta tính giá trị exp bằng cách lấy giá trị của dãy bit -BIAS. Với số không chuẩn, phần mũ bằng 0 và exp = -16382 và có dạng 0.F. Với số bình thường, exp được tính theo công thức trên và có dạng 1.F.
 - Obi dấu chấm động khi exp!= 0. exp dương thì ta dòi qua phải. Mỗi lần dịch, phần nguyên cộng thêm bit từ phần thập phân chuyển qua và phần thập phân xóa bót. Nếu đã dịch hết phần thập phân, ta thêm 0 vào phần nguyên. Sau đó, giảm exp đi 1 đơn vị. Ngược lại ta dời qua trái. Mỗi lần dịch, phần thập phân cộng thêm bit từ phần nguyên chuyển qua và phần nguyên xóa bót bit. Nếu đã dịch hết phần nguyên, ta thêm 0 vào phần thập phân. Sau đó, tăng exp lên 1 đơn vị.
 - O Chuyển phần nguyên về dạng thập phân (hệ 10)
 - Chuyển phần thập phân về dạng thập phân (hệ 10). Khi duyệt bit ở phần này, tại mỗi bit, nếu là bit 1 ta thực hiện cộng với 2^-n, với n là vị trí bit, tính từ 1.

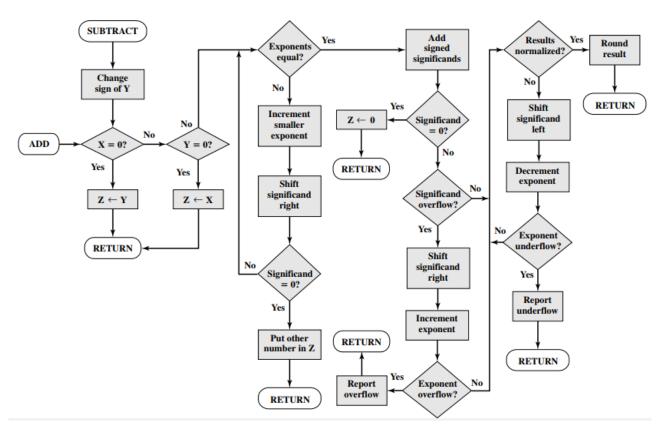
3.2.9. Chuyển đổi từ QFloat sang chuỗi ở hệ cơ số 2, 10 (string):

Ta sử dụng câu lệnh *if* trên tham số đầu vào (đầu vào sẽ được truyền vào biểu thị hệ cơ số cần được chuyển đổi ra string từ QFloat). Sau đó dựa vào đầu vào ta lần lượt gọi các hàm đã trình bày ở 3.2.7 và 3.2.8 để chuyển từ QFloat sang string.

3.3. Các nhóm toán tử:

3.3.1. Toán tử cộng:

Ta thực hiện cộng theo lưu đồ thuật toán sau đây được trích từ Chương 9, sách Computer Organization and Architecture Designing for performance (8th edition) của William Stallings.



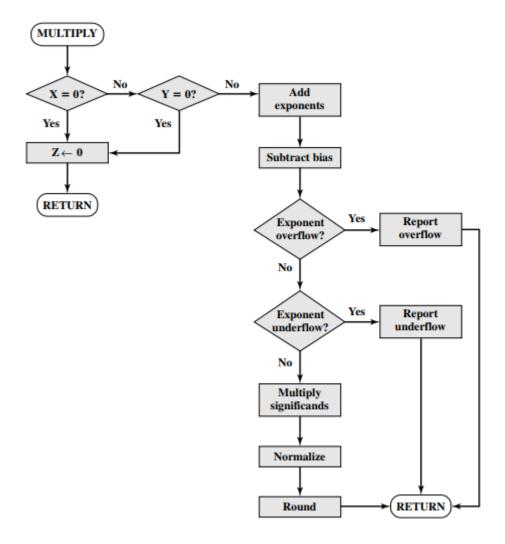
- Ta có thể tóm gọn sơ đồ trong 4 bước chính:
 - $\circ~$ Kiểm tra phép cộng với NaN, Infinity và 0.
 - Điều chỉnh phần mũ cho 2 số bằng nhau.
 - O Thực hiện cộng có dấu phần định trị của 2 số.
 - Chuẩn hóa kết quả (normalize result).

3.3.2. Toán tử trừ:

Ta thực hiện đổi số trừ sang số đối của nó (thực hiện đối dấu) và thực hiện phép cộng (vì a-b=a+(-b)).

3.3.3. Toán tử nhân:

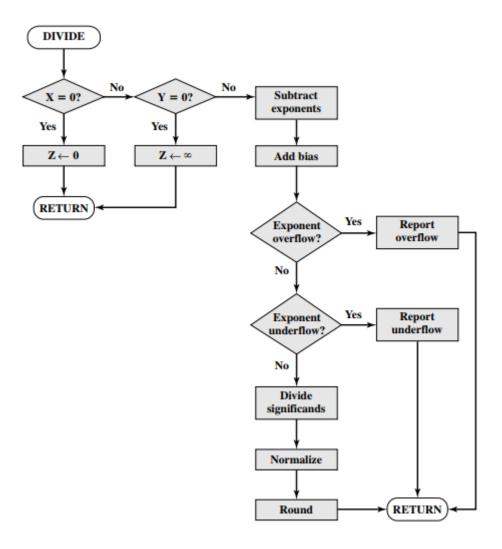
Ta thực hiện nhân theo lưu đồ thuật toán sau đây được trích từ Chương 9, sách Computer Organization and Architecture Designing for performance (8th edition) của William Stallings.



- Ta có thể tóm gọn sơ đồ trong 4 bước chính:
 - o Kiểm tra phép nhân với NaN, Infinity và 0.
 - Tính toán phần mũ
 - Thực hiện nhân phần định trị
 - Chuẩn hóa kết quả (normalize result).

3.3.4. Toán tử chia:

Ta thực hiện chia theo lưu đồ thuật toán sau đây được trích từ Chương 9, sách Computer Organization and Architecture Designing for performance (8th edition) của William Stallings.



- Ta có thể tóm gọn sơ đồ trong 4 bước chính:
 - Kiểm tra phép chia với NaN, Infinity và 0.
 - Tính toán phần mũ
 - O Thực hiện chia phần định trị
 - Chuẩn hóa kết quả (normalize result).

4. PHẠM VI BIỂU DIỄN CỦA CÁC KIỂU DỮ LIỆU ĐÃ THIẾT KẾ

4.1. Kiểu số nguyên lớn QInt 128 bit:

Do lưu trữ QInt trên 128 bit, nên phạm vi biểu diễn của QInt là từ -2^{127} đến $2^{127} - 1$.

4.2. Kiểu số chấm động chính xác cao:

Kiểu QFloat được lưu trữ bởi 1 bit dấu (s), 15 bit phần mũ (E), 112 bit phần định trị (F). Ta xét:

- Số chuẩn:
 - O Min (+): $1.[112 \text{ bit } 0] \times 2^{-16382} = 2^{-16382}$

O Max (+): 1.[112 bit 1]
$$\times 2^{16383} = (2^0 + 2^{-1} + 2^{-2} + ... + 2^{-112}) \times 2^{16383}$$

= $(2 - 2^{-112}) \times 2^{16383}$
= $2^{16384} - 2^{16271}$

- O Min (-): -1.[112 bit 1] $\times 2^{16383} = -(2^{16384} 2^{16271})$
- o Max (-): -1.[112 bit 0] $\times 2^{-16382} = -2^{-16382}$
- Số không chuẩn:
 - O Min (+): $0.[111 \text{ bit } 0]1 \times 2^{-16382} = 2^{-112} \times 2^{-16382} = 2^{-16494}$

$$0 \quad \text{Max (+): } 0.[112 \text{ bit } 1] \times 2^{-16382} = (2^{-1} + 2^{-2} + ... + 2^{-112}) \times 2^{-16382}$$
$$= (1 - 2^{-112}) \times 2^{-16382}$$
$$= 2^{-16382} - 2^{-16494}$$

- O Min (-): $-0.[112 \text{ bit } 1] \times 2^{-16382} = -(2^{-16382} 2^{-16494})$
- o Max (-): $-0.[111 \text{ bit } 0]1 \times 2^{-16382} = -2^{-16494}$
- Vậy phạm vi biểu diễn của QFloat là:

$$\left[\begin{array}{c} -(2^{16384}-2^{16271}); \, -2^{-16382} \end{array} \right] \cup \left[\begin{array}{c} -(2^{-16382}-2^{-16494}); \, -2^{-16494} \end{array} \right] \cup \left\{ \begin{array}{c} 0 \end{array} \right\} \cup \left[\begin{array}{c} 2^{-16494}; \, 2^{-16382}-2^{-16494} \end{array} \right]$$

$$\cup \left[\begin{array}{c} 2^{-16382}; \, 2^{16384}-2^{16271} \end{array} \right]$$

- Ngoài ra QFloat còn có dạng Infinity / -Infinity (phần mũ toàn bit 1, phần trị bằng 0)
- Nếu vượt ra ngoài các khoảng trên, thì QFloat sẽ biểu diễn dạng số báo lỗi NaN (phần mũ toàn bit 1, phần trị khác 0).

5. GIAO DIỆN NGƯỜI DÙNG

Giao diện được thiết kế trên C++/CLI support và .NET Framework 4.5.1. Chính vì thế, để chạy chương trình, cần cài đặt 2 components này (từ Microsoft Visual Studio 2017 về sau).

5.1. Chế độ QInt:

- Người dùng có thể nhập đầu vào ở hệ Bin, Dec hay Hex.
 - o Ở chế độ Bin, các phím nhập số sẽ bị vô hiệu hóa ngoại trừ 0 và 1.
 - $\circ~$ Ở chế độ Dec, các phím số sẽ được mở từ 0 đến 9.

- o Ở chế độ Hex, các phím số và phím chữ (A đến F) sẽ được mở.
- Các phím thực hiện chức năng sẽ được mở dù ở hệ cơ số nào.
- Phím +/- để biểu thị số âm hoặc dương ở hệ Dec.

5.2. Chế độ QFloat:

- Tương tự như QInt, người dùng có thể nhập đầu vào ở hệ Bin, Dec.
 - o Ở chế độ Bin, các phím nhập số sẽ bị vô hiệu hóa ngoại trừ 0 và 1.
 - o Ở chế độ Dec, các phím số sẽ được mở từ 0 đến 9.
- Các phím chức năng chỉ được mở phép cộng, trừ, nhân, chia (không có phép xoay, dịch, AND, OR, XOR, NOT,... như QInt).
- Phím +/- để biểu thị số âm hoặc dương ở hệ Dec.
- Phím . cũng được mở để biểu diễn số thực.

6. GIAO DIỆN CHƯƠNG TRÌNH ỨNG VỚI TEST CASE

6.1. Tham số dòng lệnh:

6.1.1. Kiểu dữ liệu QInt:

Kiểu dữ liệu QInt với 169 testcase bao gồm các yêu cầu, trường hợp như yêu cầu đề bài.



Input:

```
10 16 593851372216659226911839030355
10 16 -84828294819357048336194926778712
10 16 849298088127969977958483909
10 16 01165609898348637686922267031218
10 16 -65922773237475283722582336666073
10 16 -103003749854139019267895669715
10 16 607624685789327548934
10 16 343502432054724620814091926788
10 2 82686388520758526
10 2 15229005931757525
10 2 -64820362582556349421896059723809
10 2 1268975999699853
10 2 7714449106204895429
10 2 -547670597121268234285953806085
10 2 1146472
16 2 168B9E457FA3A0A
16 2 DF37B291E5A31CC9AF6040
16 2 BDB31991E67DFC988F
16 2 41E7B4CAB9BF6B0941DC5BB57D69AB
16 2 EF988295C2B33E7D81B8183233976D
16 2 AC2E04CEAAE3116A18287341BED80F
16 2 DE935ADD23506597D
16 2 030321
16 10 759480D203675C6D6E07E72FBD300A
16 10 344CF46BD7707150C
16 10 B53B08BE2F1B77DF8AE7146F738EDD
16 10 323A6C9AF795C4F6DA870957CDEC25
16 10 7429FF8F822CD00
16 10 46663E8
10 -2608479417 + -22338170235325255170379
10 67051 + 7441
10 453337030 + 8597
10 4529098710012059889 + 5871
16 9D851758BE0F24CDE098BD0FD4FE08 + 040149A737E212102CB9FBB29C92D8
16 9CC4C5C75E61A4B + C39821202B37680B2B
16 89CDBEE207A7177DA5F33F77CE4EAC + 713BB2777876DD1F824EFF9CC74ED0
16 9C448 - A8E8A44EFC7D8F1A4571F
16 9C448 - A0E8A44EFC7D8F1A4571F
16 A13AE7E22CAD058D30A8CAE970A8D5 - BA44D264E057CF210D46CD343B4795
16 C449B5966D1C9F3994946BFC405F5F - 2A0BFA8C735DFCDF3D4BD841
16 E70F1BFC433C9AD3E67AF21ADFC7D6 - 042932CF6C664FBA426F0B3154486F
16 F482010086824FEE461A3C14A - 3213E98CC766B0EE5DCC07E
```

10 778269483384971988474039882058 * 9450413062241862464166546321813

10 621077415018341905 * 3227

```
7459301674149 * 7346
16 7495816208A3EBC0CE872B65A3E845 * 9B1BA8EDC69155F1F3C6475990369E

16 3A3E57C0E43DA770AF764AC585CE75 * F066BB2CE94BCBE39276E79FBDA683

16 B9CB85F236B5AF5FD129291E7C9FA7 * 760E1F199060034541D4FA18F7706E

16 123F0F0F05B8229E0929055 * EB5FA0B260595E877B
16 992CE67D0594A6A * E146C7063B8172E346F82124985375
16 B8018EDF5E388 / A09B
16 2EC2F624754EE / F5CA
10 36446868780624476630340083329890 < 3191122320557494
10 -35648493304803 < -35648493304803
10 698857167199307749642781 < 1919
10 36446868780624476630340083329890 > 3191122320557494
10 -35648493304803 > -35648493304803
10 839550102901849678702 > 5856
2 001100111000010000010010 > 0110
10 -35648493304803 <= -35648493304803
10 1146472 <= 1859
16 DBDEA7 <= 7A91
10 -35648493304803 >= -35648493304803
```

```
10 36446868780624476630340083329890 == 3191122320557494
10 -35648493304803 == -35648493304803
10 -35648493304803 != -35648493304803
10 28736988017724675442080720333982 & 042499842308284840953597276
10 79746049771611191327920338690 & -366356256600176637495
10 7876617499922705901400642994 & -5324911307438252005796
16 B55625B9B285398204F9AE95AADD25 & E9739349374012035
16 BAFB91D60DC94 & 61E1
16 9B79FA0B7A8CBD81B0BB3CFD10FB1F & C680115C2FD62315DC8C9
16 46663E8 & 68FB
10 -60024762915336250482099137 | 7741143406189558483041
10 830308067922260789 | 71170691908165983228547378458460
10 7554760384 | 1377
10 80764463752 | 5680
16 E0AAE7F0205714475707CDBEEF | B0ED697A738954261EF2436A5DCC
16 F2 | 3559
10 03362910865110582745817367724014 ^ 4211663939788632886502868044476
10 858380 ^ 1425
10 -22611533712059868415925978809905 ^ 2889322464545720840180
10 72409067473542248638207766254906 ^ -5944575148319264882666945777
16 594D61FA8FA78A53653949C0D52093 ^ 6EEBA81BEC0A99D8514A678BD82FBF
16 754BCC5AF7285E65229CF3FDB92E12 ^ 8B9AA23366E0A0F046626
16 51D800C155CF840C92A7 ^ 55CSR764CC75CCA4AD73057F8445590
16 61D4800C155CE340C92A7 ^ 55C5B764CC75C04A6D7957FF44A590
```

```
10 ~ 8693247699586933108540
16 ~ 9FE8454F5FF61EC181F6986F9B9E0B
 16 ~ F1D351858F91488E70DA74EE717563
 16 ~ C9AD1C5E6165FEE2CD
 16 ~ CF55F2913C15FEBB85E8AF27AD36
16 ~ IC75C2053434570DA13BAEC3C
16 ~ 47B5449092CF567AAC38AE4BD6F831
 148
 10 23264078186954652670106142 << 8
 10 -48090253898717506892192568220146 << 5
 10 3036 << 9378
 10 6172960057076055041298954 << 5200
 16 D49ECDFBD4CF3168F1CC739 << 9
 16 FC893FC29DB2B01DC5EB7929375 << 2
16 46C5E959FB08A47CE8D17FD785E6 <<
 10 -6529624872154135214755416638567 >> 67
 10 42171454621913342120647207 >> 2
 16 045A15E6B27DB33B0AF210E2103A13
 2 11000001011110110000010000 >> 11
 10 37053621 rol 45
```

Output:

```
-5916556177087
67744706553877466985193610638133488104
-80814234504046499283663537294945540079
132816630614774637197058216469153342572
5685379816107727195046084
FC5380D4780BBFBAE920DE3900
73AAE6BAF70B37315
E198AFCBOEDEF6AB4A41A21
D7E3632E5FE6A2FB3CA5484501223D4E
A38B3EB51C53B36A4E500CE5EF70F64B
6B2D607DE47AC8AFBF3B91023F52FB13
DEDCFC1C2500F54BDF269BE55167B6F9
369F8FCD308CF2312C4226
789BB822D74DF7772816165
77ED657828A3CFD828A6B0C53
FFFFFBD150FF21269C825F2AB14466A8
2BE85DD9C9F5A2DF7C147C5
EB649E8E3029287A6D1D376B2
FFFFFCBFF0080D5F005816ECE3650627
FFFFFFEB32D4B8BB30F342F636CCC2D
20F07C88644F366206
455EA8D5A63ABD22C17E5BD04
110000001100100001
```

100 False 101 True 102 True

```
610510735270684054270771598890709002
  60298491715446576396
  941003086245820458176707444742524637
  260799827285308174746992026210659365
  523156398392134912
  -22338170235327863649796
 453345627
  4529098710012065760
 A18660FFF5F136DE0D52B8C27190E0
C3A1ED6C87AD4E2576
  FB097159801DF49D28423F14959D7C
  FFE6F6157D4C55366C2361FDB5356140
  C449B56BE12202C5D6878CBEF486FE
  E2E5E92CD6D64B19A40BE6E98B7F67
  F44FED16F9BAE93D57BC700CC
  -87468726937417317160128490266755011319
  160146180623295112916053612743908341626
  97335843693757086172644204564906825960
69 -2846445634787099864752
70 2004216818264189327435
  -28464456347870998647520687562909943278
  54796030098298554
  D026AFF5399A31241353854EF1F7E896
  BC168BAA6419CA6F9D012CEEEEB983DF
  96D9EDFC4574ECF9C5BAFB4D5186A9C2
  F3076FA8A3B0F68F780436ABCBE2BD7
```

```
D9CFF625C123722D289892442DF6072
11421332408924128
4294266168
46FDFE4B5
9CC2
1254CC23C6
30B446F0D
False
False
False
True
False
True
True
True
True
False
```

```
True
104
  True
  False
  False
 True
 False
109
 False
 42320392361246331172621340
  79746049765264317284247732480
  7876616909537935147669668880
  88200308214000025
  4080
 808010180B9203110C809
 60E8
 118
  -60021799839497755467842433
  71170691908166775959150246346621
  7554760673
 80764468920
 BOEDEBFFF3A957365FF747EFFFEF
124
 126
 45151412705704924082179923600082
 859293
 -22611533710074638017639535045061
  -72413735342744399604525796288971
 37A6C9E163AD138B34732E4B0D0F2C
  754BCC5AFA91F44614F2F9F2ED4834
  55C5B764CA68884AAC2C99CB483737
 AEF57773BD36440B6F69DB3B77C93D
138
  -8693247699586933108541
  FF6017BAB0A009E13E7E0967906461F4
140
  FF0E2CAE7A706EB7718F258B118E8A9C
 FFFFFFFFFFFFF3652E3A19E9A011D32
141
  FFFF30AA0D6EC3EA01447A1750D852C9
 FFFFFFFE38A3DFACBCBA8F25EC4513C3
  FFB84ABB6F6D30A98553C751B42907CE
145
 5955604015860391083547172352
149
  -1538888124758960220550162183044672
 1A93D9BF7A99E62D1E398E7200
 3F224FF0A76CAC07717ADE4A4DD4
 46C5E959FB08A47CE8D17FD785E600
  -44246459200
  10542863655478335530161801
 22D0AF3593ED99D85790871081D09
 160
 110000010111101
  1303708388502559260672
 169 -407457
```

6.1.2. Kiểu dữ liệu QFloat:

Kiểu dữ liệu QFloat với 127 testcase bao gồm các yêu cầu, trường hợp như yêu cầu đề bài.

```
☑ C:\Windows\$ystem32\cmd.exe
Microsoft Windows [Version 10.0.18363.720]
(c) 2019 Microsoft Corporation. All rights reserved.

C:\Users\trong\OneDrive - VNU-HCMUS\Learning HCMUS\4th Semester\Kiến trúc mấy tính & Hợp ngữ\18120197_18120230_18120336\Source\CommandLineArgument\Rele ase>18120197_18120230_18120336 input_QFloat.txt output_QFloat.txt 2
```

Input:

10 5.375 - -23.40625

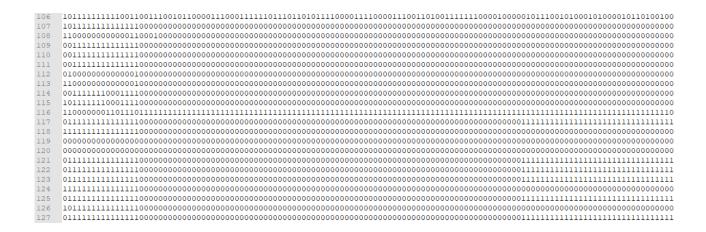
```
0.1
1.199999999999999556
10 2
10 2 26.499999986733027
10 2 2789394.743038
10 2 0.00000000000000002749
10 2 -73849.26489
-0.00000000038243
5673894384748578493840380480930094
10 2
10 2
10 2 0.000000000000000000542101086242752217003726400434970855712890625
10 2
-0.1234
10 2 5.375
10 2 -23.40625
10 2 28.78125
10 679348.274 + 7864789590349.283
10 -679348.274 + 7864789590349.283
10 679348.274 + -7864789590349.283
10 -679348.274 + -7864789590349.283
10 0 + 0
10 0.0346378483 - 63479834900.348934
10 0.0346378483 - -63479834900.348934
10 -0.0346378483 - -63479834900.348934
10 -0.0346378483 - 63479834900.348934
10 1111122222333334444455555.5555554444333332222211111 - 1111122222333334444455555.5555544444333332222211111
10 -1111122222333334444455555.12345 - 5555544444333332222211111.54321
10 9876543212345678987654321.123456789 - 12345678987654321.987654321
```

```
10 679348.274 * 7864789590349.283
10 -679348.274 * 7864789590349.283
10 679348.274 * -7864789590349.283
10 -679348.274 * -7864789590349.283
10 5384.378 * 0.005
10 -11111222233333.555566667777 * -0.000000000000000542101086242752217003726400434970855712890625
.
10 679348.274 / 7864789590349.283
10 67476584895.248 / 27730.23
10 6479394.483 / 0.05
10 -3748.8348830 / -0.002
10 0 / 3.38
10 -1 / 3
10 0 / 123.456
10 -1 / -4
10 26.8 / -7.6
```

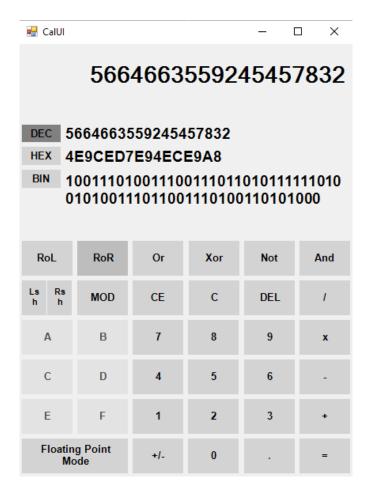
Output:

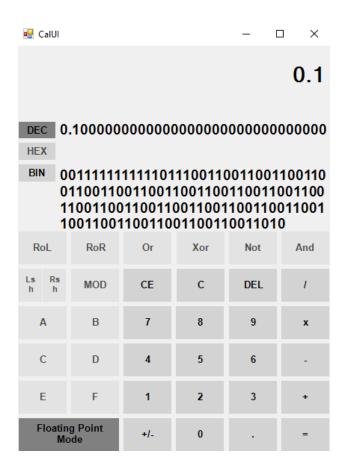


```
123456789.123449999999999999999999555526323815512763130974871472744780476205050945281982421875
-123456789.1234499999999999999999999559526323815512763130974871472744780476205050945281982421875
-23.40625
7864790269697.55699999999999999999998576984648612775430365218198858201503753662109375
7864788911001.009000000000000000001558540622947912623885713401250541210174560546875
-7864788911001.00900000000000000001558540622947912623885713401250541210174560546875
-7864790269697.5569999999999999999998576984648612775430365218198858201503753662109375
0.299999999999999999999999999999990370350278063820734720110287075363407309491758923059023800306022167205810546875
-63479834900.31429615170000000000004884482575351345445824335911311209201812744140625
63479834900.38357184830000000000000382368290675894062502493397914804518222808837890625
63479834900.31429615170000000000004884482575351345445824335911311209201812744140625
-63479834900.38357184830000000000000382368290675894062502493397914804518222808837890625
-6666666666666666666666666666666599996387958526611328125
0.2999999999999999999990000000000029767434726102847930917622623817986142469305121949219028465449810028076171875
5342931233576952463.1875419999999965342567520565353333950042724609375
-5342931233576952463.18754199999999965342567520565353333950042724609375
-5342931233576952463.1875419999999965342567520565353333950042724609375
5342931233576952463.18754199999999965342567520565353333950042724609375
26.9218900000000000000000000000000000001097502734388732674279067366449676342032848452845428255386650562286376953125
0.00006023405636753620415628440786046798466299145992566307829264289848280220661923346271195145451571306693949736654758453369140625
0.0000000863784423214085602168857836998746029429333236259749582884449976804696723588121188633901816056326339321458362974226474761962890625
2433322.2225437005030250380180763014327789838806834072358099840815981451669358648359775543212890625
129587889.65999999999999999999999898980727029261462192322174047376392991282045841217041015625
1874417.44150000000000000000000000084010530962416873229127445288355602315277792513370513916015625
-0.333333333333333333333333333333333317283917130106367891200183811792272345515819598205098373000510036945343017578125
-3.526315789473684210526315789473684514620517534826713640417569881830629242858154981377083458937704563140869140625
```



6.2. Giao diện người dùng:





7. ĐÁNH GIÁ MỨC ĐỘ HOÀN THÀNH

Kiểu dữ liệu / Công việc chính	Công việc	Mức độ hoàn thành		
	Nhập, xuất	100%		
	Chuyển đổi giữa hệ cơ số a sang hệ cơ số b (với a, b thuộc {2, 10, 16})	100%		
	Toán tử cộng, trừ, nhân, chia, chia lấy dư	100%		
QInt	Các toán tử so sánh và gán (<, >, <=, >=, ==, !=, =)	100%		
	Các toán tử AND, OR, XOR, NOT	100%		
	Toán tử dịch trái, dịch phải, xoay trái, xoay phải	100%		
QFloat	Nhập, xuất	100%		

	Chuyển đổi giữa hệ cơ số a sang hệ cơ số b (với a, b thuộc {2, 10})	100%
	Toán tử cộng, trừ, nhân, chia	100%
Giao diện	Thiết kế giao diện Calculator ở 2 chế độ QInt và QFloat	100%
Tổng thể đồ án		100%

8. TÀI LIỆU THAM KHẢO

- [1] Slides bài giảng Kiến trúc máy tính và hợp ngữ, Thầy ThS Phạm Tuấn Sơn, trường Đại học Khoa học Tự nhiên.
- [2] Quadruple-precision floating-point format: https://en.wikipedia.org/wiki/Quadruple-precision-floating-point-format
- [3] Floating-Point Arithmetic, A. Yavuz Oruç Professor, UMD, College Park: https://user.eng.umd.edu/~yavuz/teaching/courses/enee446/lecture%20notes/lecture4467N8
 N9.ppt.pdf
- [4] Computer Organization and Architecture Designing for performance (8th edition), William Stallings.
- [5] Visual C++ Calculator Tutorial, DJ Oamen: https://youtu.be/zBBe9oKU7jk